Tema 5: Prioritetsköer

Henrik Thulin `heth7132`
Mattin Lotfi `malo5163`

14 februari 2016

1 Prioritetsköer

1.1 Förändringar

1.1.1 Konstruktorn

```
private int d = 2;

public DHeap( )
{
    this( 2 );
}

public DHeap( int d )
{
   if (d < 2)
        throw new IllegalArgumentException("Must be atleast 2");
```

```
    this.d = d;
    currentSize = 0;
    array = (AnyType[]) new Comparable[ DEFAULT_CAPACITY + 1 ];
}
```

Konstruktorn med parameter initierar, istället för kapacitet, antalet barn heapen ska ha. Skulle barnantalet vara under det tillåtna värdet av 2 kastas en *IllegalArgumentException* exception.

Konstruktorn utan parameter använder den andra konstruktorn med parameter för att, enligt instruktion, skapa en *2-heap*.

1.1.2 parentIndex

```
public int parentIndex(int i) {

    if (i < 2)
        throw new IllegalArgumentException();

    return (i + (d - 2)) / d;
}
```

Räknar ut förälder till node X. Skulle X vara 1 eller felaktigt värde kastar en *IllegalArgumentException*. Positionen beräknas genom att X justeras för förskjutning och sedan delas med antalet barn per nod.

1.1.3 firstChildIndex

```
public int firstChildIndex(int i) {

    if (i < 1)
        throw new IllegalArgumentException();

    return i * d - (d - 2);
}
```

Räknar ut första barnet till nod X. Skulle X ha ett felaktigt värde kastas en *IllegalArgumentException*. Positionen räknas ut genom att gångra noden med antal barn per nod och sedan justera efter den förskjutning som sker med antalet barn.

1.1.4 Size

```
public int size() {
    return currentSize;
}
```

Returnerar antalet använda positioner

1.1.5 get

```
public AnyType get(int i) {
    return array[i];
}
```

Returnerar element från specifik position

1.1.6 insert

```
public void insert( AnyType x )
{
    if( currentSize == array.length - 1 )
        enlargeArray( array.length * 2 + 1 );

    int hole = ++currentSize;

    if (hole > 1) {

        int parentHole;

        do {
```

```
            parentHole = parentIndex(hole);

            if (x.compareTo(array[parentHole]) < 0) {
                array[hole] = array[parentHole];
                hole = parentHole;
            }
            else
                break;
        }
        while (parentHole != 1);

    }

    array[ hole ] = x;
}
```

Rutinen för percolate up har ändrats. Nu används *parentIndex* metoden, som är bättre anpassad för X barn per nod, för att räkna ut förälder.

1.1.7 percolateDown

```
private void percolateDown( int hole )
{
    int smallest =  firstChildIndex(hole);

    if (smallest <= currentSize) {

        for (int p = smallest + 1; p < smallest + d && p <= current
            if (array[smallest].compareTo(array[p]) > 0)
                smallest = p;

        if (array[hole].compareTo(array[smallest]) > 0) {

            AnyType tmp = array[hole];
            array[hole] = array[smallest];
            array[smallest] = tmp;

            percolateDown(smallest);
        }
    }
}
```

Använder den nya *firstChildIndex* metoden för att räkna ut första barn för nod X. Kontrollerar om något av barnnoderna är mindre till "Hole". Om så, så byter de plats. Sen anropas metoden med samma element fast på nya positionen. Så fortsätter det tills inget av barnen längre är av ett lägre värde eller tills det inte finns några undernoder.

2 Frågor

2.1 Fråga 1

Varför ska trädet vara vänsterbalanaserat?

2.2 Fråga 2

Hur skiljer sig en prioritetskö från en vanlig kö/queue?